

Lenguaje AMPL.

Pedro Luis Luque

Marzo de 2000. Versión: 2a

Tabla de Contenido

1 Lenguaje AMPL.	1
1.1 Reglas léxicas de AMPL.	3
1.2 Los elementos de un conjunto.	4
1.3 Expresiones que indexan y subíndices.	4
1.4 Expresiones aritméticas, lógicas y de conjuntos.	5
1.5 Declaraciones de elementos del modelo.	8
1.6 Especificación de datos.	12
1.7 Comandos del lenguaje.	20
1.8 Optimizadores.	22
2 Entorno AMPL para Msdos.	23
3 Entorno AMPL para Windows: AMPL Plus.	24
3.1 Componentes de AMPL Plus.	25
3.2 Resolución de problemas en AMPL Plus.	27

1 Lenguaje AMPL.

AMPL es un lenguaje de modelado algebraico para programación matemática: un lenguaje capaz de expresar en notación algebraica problemas de optimización tales como los problemas de programación lineal. Veamos un pequeño ejemplo.

Ejemplo 1.1. Una compañía fabrica tres productos, P_1 , P_2 y P_3 , que precisan para su elaboración dos materias primas, M_1 y M_2 . Las disponibilidades semanales de estas materias son 25 y 30 unidades, respectivamente.

El beneficio neto que proporciona cada unidad de producto, así como las unidades de materia prima que necesita para su elaboración, vienen dados en la siguiente tabla:

	P_1	P_2	P_3
M_1	1	2	2
M_2	2	1	3
Beneficio (u.m.)	2	6	3

Planificar la producción semanal de forma que se maximice el beneficio.

Solución:

Sean x_1, x_2, x_3 (x_i) la cantidad producida de P_1, P_2, P_3 respectivamente ($P_i, i = 1, 2, 3$).

El problema a resolver sería el siguiente:

$$\begin{aligned} \max \quad & z = 2x_1 + 6x_2 + 3x_3 \\ \text{s.a.} \quad & x_1 + 2x_2 + 2x_3 \leq 25 \\ & 2x_1 + x_2 + 3x_3 \leq 30 \\ & x_1, x_2, x_3 \geq 0 \end{aligned}$$

El modelo (modelo+datos) escrito en AMPL del ejemplo (1.1) fue el siguiente:

```
# FABRICACION DE 3 PRODUCTOS CON 2 MATERIAS PRIMAS
# VARIABLES DE DECISION Y RESTRICCIONES NO NEGATIVIDAD
var x1 >= 0;
var x2 >= 0;
var x3 >= 0;
# FUNCION OBJETIVO DEL MODELO
maximize z : 2*x1 + 6*x2 + 3*x3;
# RESTRICCIONES DEL MODELO
subject to restriccion1 : x1 + 2*x2 + 2*x3 <= 25;
subject to restriccion2 : 2*x1 + x2 + 3*x3 <= 30;
```

Tabla 1: Modelo básico del ejemplo.

1.0.1 Ejemplo en AMPL.

La gran potencia del lenguaje AMPL está en separar el **modelo** en sí por un lado y por otro los **datos** particulares del problema concreto. Para entender esto mejor escribimos el problema del ejemplo 1.1 desde este punto de vista.

El modelo general con n productos y con m materias primas podríamos escribirlo de la siguiente manera:

$$\begin{aligned} \max \quad & z = c_1x_1 + c_2x_2 + \dots + c_nx_n \\ \text{s.a.} \quad & a_{11}x_1 + \dots + a_{1n}x_n \leq b_1 \\ & a_{21}x_1 + \dots + a_{2n}x_n \leq b_2 \\ & \vdots \\ & a_{m1}x_1 + \dots + a_{mn}x_n \leq b_m \\ & x_1, x_2, \dots, x_n \geq 0 \end{aligned}$$

O de forma más abreviada:

$$\begin{aligned} \max \quad & z = \sum_{j=1}^n c_jx_j \\ \text{s.a.} \quad & \sum_{j=1}^n a_{ij}x_j \leq b_i, \forall i = 1, \dots, m \\ & x_j \geq 0, j = 1, \dots, n \end{aligned} \tag{1}$$

En este problema general los valores de c_j , a_{ij} y b_i son datos del problema concreto y la estructura del problema (1) sería el modelo general.

Esto en AMPL se escribiría de la forma siguiente en dos ficheros. En el fichero del **modelo** aparecería:

```

# MODELO: EJEMPLO1.MOD
# FABRICACION DE n PRODUCTOS CON m MATERIAS PRIMAS
# PARAMETROS DEL MODELO
param n >=0, integer;
param m >=0, integer;
# CONJUNTOS DE INDICES
set PRODUCTOS := 1..n;
set MPRIMAS := 1..m;
# VARIABLES DE DECISION Y RESTRICCIONES NO NEGATIVIDAD
var x {j in PRODUCTOS} >= 0;
# MAS PARAMETROS DEL MODELO
param c {i in PRODUCTOS};
param b {j in MPRIMAS};
param a {(i,j) in {MPRIMAS,PRODUCTOS}};
# FUNCION OBJETIVO DEL MODELO
maximize z : sum {j in PRODUCTOS} c[j]*x[j];
# RESTRICCIONES DEL MODELO
subject to restriccion {i in MPRIMAS} :
    sum {j in PRODUCTOS} a[i,j]*x[j] <= b[i];

```

Tabla 2: Modelo general del ejemplo.

Y el fichero de **datos** para nuestro ejemplo particular sería:

```

# DATOS: EJEMPLO1.DAT
param n := 3;
param m := 2;
param c:=
    1 2
    2 6
    3 3;
param a : 1 2 3:=
    1    1 2 2
    2    2 1 3;
param b:=
    1 25
    2 30;

```

Tabla 3: Datos para el ejemplo.

Podríamos escribir un fichero de datos con valores diferentes y resolverlo junto al modelo general (por ejemplo con $n = 4$ y $m = 5$).

1.1 Reglas léxicas de AMPL.

Los modelos AMPL envuelven **variables, restricciones, y objetivos**, expresados con la ayuda de **conjuntos y parámetros**. A todos se llama **elementos del modelo**. Cada elemento del modelo tiene un nombre alfanumérico (una cadena de uno o más letras, dígitos, y caracteres de subrayado): **x1, z, restriccion1**. **Nota:** las letras mayúsculas son distintas de las letras minúsculas en AMPL.

Existen dos tipos de constantes:

- **Constantes numéricas:** un signo opcional, una secuencia de dígitos que pueden contener un punto decimal, y un exponente opcional que comienza con una de las letras: **d,D,e,E**, como en 1.23D-45. Toda la aritmética en AMPL tiene la misma precisión (sobre la mayoría de las máquinas tiene doble precisión).
- **Constantes literales** son cadenas delimitadas por una comilla simple o por dobles comillas. Si la comilla simple forma parte de la constante literal debe aparecer dos veces seguidas (igual ocurre con la doble comilla).

Cada línea de instrucciones debe ir finalizada con **un punto y coma (;)**.

Los comentarios comienzan con **#** y se extienden hasta el final de la línea, o se pueden delimitar entre **/*** y ***/**, en cuyo caso pueden extenderse más de una línea.

1.2 Los elementos de un conjunto.

Un conjunto contiene cero o más elementos o miembros, cada uno de los cuales es una lista ordenada de una o más componentes. Cada miembro de un conjunto debe ser distinto. Todos los miembros deben tener el mismo número de componentes; a este número común se le conoce como **dimensión del conjunto**.

Un conjunto explícitamente se escribe como una lista de miembros separada por comas, colocados entre llaves: “{ ... }”. Si el conjunto es de dimensión uno, los miembros son simplemente constantes numéricas o constantes de cadena, o cualquier expresión cuyo valor sea un número o una cadena:

```
"a", "b", "c"
{1, 2, 3, 4, 5, 6, 7, 8, 9}
{t, t+1, t+2}
```

Para un conjunto multidimensional, cada miembro debe escribirse como una lista separada por comas entre paréntesis:

```
("a", 2), ("a", 3), ("b", 5)
{(1, 2, 3), (1, 2, 4), (1, 2, 5), (1, 3, 7), (1, 4, 6)}
```

El valor de un miembro numérico es el resultado de redondear su representación decimal por un valor real de precisión limitada. Los miembros numéricos que parecen diferentes pero que al ser redondeados al valor de precisión limitada son el mismo, tales como 1 y 0.01E2, son considerados iguales.

1.3 Expresiones que indexan y subíndices.

Muchos elementos de AMPL pueden definirse como colecciones indexadas sobre un conjunto; los miembros individuales son seleccionados al añadir un **“subíndice”** al nombre del elemento. El rango de posibles subíndices es indicado por **una expresión que indexa** en la declaración del modelo. Operadores de reducción, tales como **sum**, también usan expresiones que indexan para especificar conjuntos sobre los que las operaciones son iteradas.

Un subíndice es una lista de expresiones simbólicas o numéricas, separadas por comas y encerradas entre corchetes, como en:

```
demandas[i]
costos[j,p[k], "0-"]
```

Cada expresión con subíndices debe evaluar a un número o a un literal. El valor resultante o secuencia de valores debe dar un miembro de un conjunto de índices unidimensional o multidimensional.

Una expresión que indexa es una lista de expresiones de conjunto separadas por comas, seguida opcionalmente por dos puntos “:” y una expresión lógica, todo encerrado entre llaves:

- Expresiones que indexan:

```

{ lista de expresiones de conjuntos }
{ lista de expresiones de conjuntos : expresión lógica }

```

- lista de expresiones de conjuntos:

```

expresión de conjuntos
miembro-ciego in expresión de conjuntos
lista de expresiones de conjuntos , lista de expresiones de conjuntos

```

Cada expresión de conjunto puede ser precedida por un **miembro ciego** y la palabra clave **in**. Un miembro ciego para un conjunto unidimensional es un nombre no referenciado, es decir, un nombre no definido hasta ese punto. Un miembro ciego para un conjunto multidimensional es una lista separada por comas, encerrada entre paréntesis, de expresiones o nombres no referenciados; la lista debe incluir al menos un nombre no referenciado.

Un miembro ciego introduce uno o más índices ciegos (nombres no referenciados en sus componentes), cuyo **campo**, o rango de definición, comienza en la siguiente expresión de conjunto; el campo de un índice corre a través del resto de la expresión que indexa, hasta el final de la declaración usando la expresión que indexa, o hasta el final del operando que usa la expresión que indexa. Cuando un miembro ciego tiene una o más expresiones componentes, los índices ciegos del miembro ciego varían sobre una **proyección** del conjunto; es decir, ellos toman todos los valores para que el miembro ciego pertenezca al conjunto.

```

{A} # todos los elementos de A
{A,B} # todos los pares, uno de A, uno de B
{i in A, j in B} # el mismo que antes
{i in A, C[i]} # el mismo que antes
{i in A, (j,k) in D} # 1 de A y 1 (un par) de D
{i in A: p[i]>30} # todo i de A tal que p[i] sea mayor que 30
{i in A, j in C[i]: i<=j} # nota: i y j deben ser numéricos
{i in A, (i,j) in D: i<=j} # todos los pares con i en A y i,j en D
# (mismo valor de i) e i menor o igual que j

```

El argumento opcional “: **expresión lógica**” en una expresión que indexa, selecciona solamente los miembros que verifican la expresión lógica y excluye a los demás. La expresión lógica típicamente envuelve uno o más índices ciegos de la expresión que indexa.

1.4 Expresiones aritméticas, lógicas y de conjuntos. Funciones matemáticas.

En las expresiones lógicas y aritméticas de AMPL, pueden combinarse varios elementos. A partir de ahora consideraremos que una expresión que puede contener variables se representará como **vexpr**. Una expresión que no puede contener variables se denota como **expr** y algunas veces será llamada “expresión constante”, a pesar de que pueda contener índices ciegos. Una expresión lógica, representada como **lexpr**, puede contener variables sólo cuando es parte de una expresión **if** que produzca una **vexpr**. Las expresiones de conjuntos serán denotadas como **sexpr**.

Los operadores aritméticos, lógicos y de conjuntos, en orden de precedencia creciente, se muestran en la tabla 4.

Los valores numéricos que aparecen como expresiones lógicas valen **falso** (false) si es 0, y **verdadero** (true) para cualquier otro valor numérico.

1.4.1 Expresiones Aritméticas.

Las expresiones aritméticas son construidas con los operadores aritméticos usuales, con funciones de AMPL, y con operadores de reducción aritméticos como **sum**:

Nombre	Tipo	Notas
if ... then ... else	A,S	A: si no hay else , se supone else 0 S: es obligatorio else sexpr
or, 	L	o lógico
exists, forall	L	operadores de reducción lógica
and, &&	L	y lógico
<, <=, =, ==, <>, !=, >=, >	L	operadores relacionales
in, not in	L	pertenencia a un conjunto
within, not within	L	S within T significa $S \subseteq T$
not, !	L	negación lógica
union, diff, symdiff	S	symdiff es la diferencia simétrica
inter	S	intersección
cross	S	producto cartesiano
setof .. by	S	constructor de conjuntos
+, -, less	A	a less b = $\max(a - b, 0)$
sum, prod, min, max	A	operadores de reducción aritmética
*, /, div, mod	A	div cociente entero
+, -	A	más y menos unario
^, **	A	exponenciación

Tabla 4: Operadores aritméticos (A), lógicos (L) y de conjuntos (S).

```

numero
variable
expresión  op.aritmético  expresión  (+, -, less, *, /, mod, div, ^, **)
unario    expresón
función( lista de expresiones )
if lexpr then vexpr [ else vexpr ]
operador-reducción  expresón que indexa (sum, prod, max, min)
{ expr }

```

Las funciones aritméticas incorporadas en AMPL se muestran en la tabla 5 y tabla 6.

Sintaxis	Significado
Beta(a,b)	$x^{a-1}(1-x)^{b-1}/(\Gamma(a)\Gamma(b)/\Gamma(a+b)), x \in [0, 1]$
Cauchy()	$1/(\pi(1+x^2))$
Exponential()	$e^{-x}, x > 0$
Gamma(a)	$x^{a-1}e^{-x}/\Gamma(a), x \geq 0, a > 0$
Irاند224()	Uniforme entera en $[0, 2^{24})$
Normal(μ, σ)	$N(\mu, \sigma)$ varianza
Normal01()	$N(0, 1)$
Poisson(μ)	$e^{-\mu}\mu^k/k!, k = 0, 1, \dots$
Uniform(m,n)	Uniforme $[m, n)$
Uniform01()	Uniforme $[0, 1)$

Tabla 5: Funciones de generación de var. aleatorias en AMPL.

Sintaxis	Significado
abs(x)	valor absoluto
acos(x)	$\cos^{-1}(x)$
acosh(x)	$\cosh^{-1}(x)$
asin(x)	$\sin^{-1}(x)$
asinh(x)	$\sinh^{-1}(x)$
atan(x)	$\tan^{-1}(x)$
atan2(y,x)	$\tan^{-1}(y/x)$
atanh(x)	$\tanh^{-1}(x)$
ceil(x)	entero mayor más cercano
cos(x)	coseno
exp(x)	exponencial
floor(x)	menor entero más cercano
log(x)	$\log_e(x)$
log10(x)	$\log_{10}(x)$
max(x,y,...)	máximo
min(x,y,...)	mínimo
sin(x)	seno
sinh(x)	seno hiperbólico
sqrt(x)	raíz cuadrada
tan(x)	tangente
tanh(x)	tangente hiperbólica
precision(x,n)	x redondeada a n cifras significativas
round(x,n)	x redondeado a n dígitos después del punto decimal
round(x)	x redondeado al entero más cercano
trunc(x,n)	x truncado a n dígitos después del punto decimal
trunc(x)	x truncado a un entero

Tabla 6: Funciones aritméticas en AMPL.

Sobre los operadores de reducción aritmética:

- La palabra clave **sum** debe seguirle cualquier expresión que indexa. La expresión aritmética siguiente se evalúa una vez para cada miembro del conjunto de índices, y todos los valores resultantes se suman. El operador **sum** tiene menor precedencia que *, así podemos escribir:

$$\mathbf{sum} \{i \text{ in } \text{ORIG}, j \text{ in } \text{DEST}, p \text{ in } \text{PROD}\} \\ \text{costo}[i, j, p] * \text{Trans}[i, j, p]$$

representa el total de $\text{costo}[i, j, p] * \text{Trans}[i, j, p]$ sobre todas las combinaciones de orígenes, destinos y productos.

- Otros operadores aritméticos iterados son **prod** para la multiplicación, **min** para el mínimo, y **max** para el máximo. Como ejemplo, podríamos usar:

$$\mathbf{max} \{i \text{ in } \text{ORIG}\} \text{oferta}[i, p]$$

para describir la mayor oferta del producto p de todos los orígenes.

1.4.2 Expresiones Lógicas.

Las **expresiones lógicas** aparecen donde se requiera un valor: “verdadero” o “falso”. Por ejemplo, en el comando **check**, en la parte “tal que” de las expresiones que indexan (sigue a los “:”), y en

if *lexp* **then** ...

Las expresiones lógicas pueden ser de la siguiente forma (**lexpr**):

```
expr
expr oper--compara expr
lexpr oper--logico lexpr
not lexpr
miembro in sexpr
miembro not in sexpr
sexpr within sexpr
sexpr not within sexpr
exists indexado lexpr
forall indexado lexpr
{ lexpr }
```

El operador **exists**, cuando se aplica sobre un conjunto vacío, devuelve falso y el operador **forall** devuelve verdadero.

1.4.3 Expresiones de conjunto.

Las **expresiones de conjuntos** (**sexpr**) que producen conjuntos pueden tener uno de los siguientes formatos:

```
{ [miembro [ , miembro ... ] ] }
sexpr op--conjunto sexpr      (union diff symdiff inter cross)
union indexado sexpr
inter indexado sexpr
expr .. expr [ by expr ]
setof indexado miembro
if lexpr then sexpr else sexpr
( sexpr )
interval
conj--predefinido
indexado
```

Podemos ver el uso del operador **setof**, en el siguiente ejemplo:

```
ampl: set y := setof {i in 1..5} (i,i^2);
ampl: display y;
set y := (1,1) (2,4) (3,9) (4,16) (5,25);
```

1.5 Declaraciones de elementos del modelo.

La declaración de los elementos del modelo tiene el siguiente formato general:

elemento nombre [alias] [exp. indexada] [cuerpo] ;
--

Las palabras claves para los elementos del modelo AMPL pueden ser una de las siguientes:

```
set
param
var
arc
minimize
maximize
subject to, subj to, s.t.
node
```


Si se suprime el nombre del elemento se supone que es **subject to**.

Las declaraciones pueden aparecer en cualquier orden, con la excepción de que cada nombre debe estar declarado antes de ser usado.

Para las declaraciones de variables, restricciones y objetivos, se permite una forma especial de expresión indexada:

`{ if lexpr}`

Si la expresión lógica `lexpr` es verdad, entonces el resultado es un elemento simple (no indexado); en otro caso el elemento es excluido del modelo.

1.5.1 Declaración de conjuntos.

La declaración de un conjunto del modelo tiene el siguiente formato general:

```
set nombre [alias] [exp. indexada] [atributos] ;
```

en la que atributos es una lista de atributos opcionalmente separada por comas. Los cuales pueden ser (`sexpr` indica una expresión de conjuntos):

```
dimen n
within sexpr
:= sexpr
default sexpr
```

La frase `:=` especifica un valor para el conjunto; esto implica que el conjunto no será definido posteriormente en una línea de instrucciones específica para datos (`:=` y **default** son mutuamente exclusivas). El conjunto vacío se indica con: `{}`.

Existe la función `card(S)` la cual da el número de elementos del conjunto `S`.

También se pueden realizar operaciones entre conjuntos, como:

```
set A:= 1..n;
set B:= i..j by k;
set C:= A diff B;
set D:= A union B;
set E:= A inter B;
set F:= A symdiff B;
```

Se pueden definir conjuntos con infinitos elementos (**Nota:** no se puede iterar sobre conjuntos infinitos), los clásicos intervalos cerrados, abiertos o semicerrados, bien de números reales (**interval** `[a,b]`) o bien de números enteros (**integer** `[a,b]`). **Nota:** la palabra **interval** puede omitirse.

1.5.2 Declaración de parámetros.

La declaración de un parámetro del modelo tiene el siguiente formato general:

```
param nombre [alias] [exp. indexada] [atributos] ;
```

en la que atributos es una lista de atributos opcionalmente separada por comas. Los cuales pueden ser (`sexpr` indica una expresión de conjuntos):

```
binary
integer
symbolic
oprel expr
in sexpr
:= expr
default expr
```

donde “oprel” puede ser:

```
< <= = == != <> > >=
```

El atributo **in** especifica un chequeo para ver que el parámetro se encuentra en el conjunto dado.

Los parámetros indexados pueden definirse de forma recursiva. Por ejemplo:

```
param comb 'n sobre k' {n in 0..N, k in 0..n}
:= if k=0 or k=n then 1 else comb[n-1,k-1] + comb[n-1,k];
```

Infinity

es un parámetro predefinido; al igual que **-Infinity**.

1.5.3 Declaración de variables.

La declaración de una variable del modelo tiene el siguiente formato general:

```
var nombre [alias] [exp. indexada] [atributos] ;
```

en la que atributos es una lista de atributos opcionalmente separada por comas. Los cuales pueden ser (vexpr indica una expresión de variables):

```
binary
integer
>= expr
<= expr
:= expr
default expr
= vexpr
coeff [exp. indexada] restricción vexpr
cover [exp. indexada] restricción
obj [exp. indexada] objetivo vexpr
```

Las frases **>=** y **<=** especifican cotas, y la frase **:=** indica un valor inicial. La frase **default** indica los valores iniciales por defecto, cuyos valores pueden darse en una línea de instrucciones específica para datos.

Las frases **coeff** y **obj** se utilizan para la generación de coeficientes por columnas; estas especifican los coeficientes que serán colocados en la restricción indicada u objetivo indicado, el cual debe ser previamente referenciado usando **to_come**. Una frase **cover** es equivalente a la frase **coeff** pero con vexpr igual a 1.

1.5.4 Declaración de restricciones.

La declaración de una restricción del modelo tiene el siguiente formato general:

```
[subject to] nombre [alias] [exp. indexada] [ := dual--inic]
[default dual--inic] [ : expr restricción];
```

La frase opcional **:= dual--inicial** especifica un valor inicial para la variable dual (multiplicador de Lagrange) asociado con la restricción. La expresión de restricción debe estar en uno de los siguientes formatos:

```
vexpr <= vexpr
vexpr = vexpr
vexpr >= vexpr
expr <= vexpr <= expr
expr >= vexpr >= expr
```

Para permitir la generación de coeficientes por columna para la restricción, una de las **vexprs** puede tener una de las siguientes formas:

```

to_come + vexpr
vexpr + to_come
to_come

```

Los términos de esta restricción que se especifican en una declaración **var** son colocados en la posición de **to_come**.

1.5.5 Declaración de objetivos.

La declaración de un objetivo del modelo tiene el siguiente formato general:

```

maximize nombre [alias] [exp. indexada] [: expresion] ;
minimize nombre [alias] [exp. indexada] [: expresion] ;

```

y puede especificarse una expresión en una de las siguientes formas:

```

vexpr
to_come + vexpr
vexpr + to_come
to_come

```

La forma **to_come** permite la generación de coeficientes por columna, como con las restricciones.

1.5.6 Notación de sufijos para valores auxiliares.

Las variables, restricciones, y objetivos tienen una variedad de valores auxiliares asociados, a los cuales se puede acceder añadiendo al nombre uno de los siguientes sufijos dependiendo del tipo de elemento del modelo.

Sufijos para variables	
.init	valor actual inicial
.init0	valor inicial inicial (x_j^0)
.lb	cota inferior actual
.lb0	cota inferior inicial (l_j)
.lrc	costo reducido menor (.rc, $x_j \geq l_j$)
.lslack	menor holgura ($x_j - l_j$)
.rc	costo reducido: $-(z_j - c_j)$
.slack	min(lslack, uslack)
.ub	cota superior actual
.ub0	cota superior inicial (u_j)
.urc	costo reducido superior (.rc, $x_j \leq u_j$)
.uslack	holgura superior ($u_j - x_j$)
.val	valor actual (x_j)

Sufijos para restricciones	
.body	valor actual del cuerpo de la restricción ($\mathbf{A}_i\mathbf{x}$)
.dinit	valor inicial actual para la variable dual
.dinit0	valor inicial inicial para la variable dual (w_i^0)
.dual	variable dual actual (w_i)
.lb	cota inferior (rl_i)
.ldual	valor dual menor ($.dual, \mathbf{A}_i\mathbf{x} \geq rl_i$)
.lslack	holgura menor ($\mathbf{A}_i\mathbf{x} - rl_i$)
.slack	$\min(\text{lslack}, \text{uslack})$
.ub	cota superior (ru_i)
.udual	valor dual superior ($.dual, \mathbf{A}_i\mathbf{x} \leq ru_i$)
.uslack	holgura superior ($ru_i - \mathbf{A}_i\mathbf{x}$)

Sufijos para objetivos	
.val	valor actual

1.6 Especificación de datos.

Hay que tener en cuenta que:

- La lectura de datos se inicializa con el comando: “data”. Por ejemplo:

```
ampl: data diet.dat;
```

lee comandos de datos de un fichero llamado `diet.dat`.
- AMPL trata cualquier secuencia de espacios, tabuladores y caracteres de nueva línea como un solo espacio.
- El final de cualquier comando de datos se indica por un punto y coma “;”.

1.6.1 Datos en conjuntos.

Conjuntos unidimensionales.

Un conjunto simple se especifica al listar sus miembros.

```
set ORIG := SE MD BA ;
set DEST := CA CO HU AL JA MA GR ;
set PROD := plato cuchillo tenedor ;
```

Si un conjunto se ha declarado con el atributo **ordered** o **circular**, debemos listar sus miembros en orden:

```
set SEMANAS := 27sep93 04oct93 11oct93 18oct93 ;
```

- Si una cadena de la lista incluye caracteres distintos de letras, dígitos, signo de subrayado (`-`), punto, `+` y `-`, debe ser cerrado entre comillas:

```
set ALMACEN := "A&P" JEWEL VONS ;
```
- También para distinguir cuando un número queremos que sea una cadena (“+1” o “3e4”), éste debe ser encerrado entre comillas.
- Los miembros de un conjunto deben ser todos diferentes; AMPL avisaría de la existencia de elementos duplicados. Los números que tienen la misma representación en el ordenador serán considerados como iguales.

- Para una colección indexada de conjuntos, los miembros de cada conjunto de la colección se especificarán individualmente.

```

set PROD;
set AREA {PROD};

set PROD := plato cuchillo ;
set AREA[plato] := este norte ;
set AREA[cuchillo] := este oeste export ;

```

- Podemos especificar explícitamente que uno o más de esos conjuntos es vacío, al dar una lista vacía; poniendo el punto y coma justo después del operador “:=”. Si queremos que AMPL suponga que cada conjunto es vacío excepto si se especifica otra cosa, incluyendo una frase **default** en el modelo:

```

set AREA {PROD} default {};

```

En otro caso seríamos avisados de que la especificación de los miembros de un conjunto no se ha realizado.

Conjuntos de dos dimensiones.

Para un conjunto de pares, los miembros pueden especificarse de varias maneras:

```

set ORIG;
set DEST;
set LINKS within {ORIG,DEST};

```

1. lista de pares

```

set LINKS :=
(SE,CO) (SE,HU) (SE,JA) (SE,GR) (MD,CA)
(MD,CO) (MD,HU) (MD,AL) (MD,JA) (MD,GR)
(BA,CA) (BA,AL) (BA,JA) (BA,MA) ;

```

2. lista de pares, sin los paréntesis y las comas

```

set LINKS :=
SE CO SE HU SE JA SE GR MD CA
MD CO MD HU MD AL MD JA MD GR
BA CA BA AL BA JA BA MA ;

```

3. Un conjunto de pares puede especificarse en una tabla también de la siguiente forma:

```

set LINKS:  CA CO HU AL JA MA GR :=
SE      -   +   +   -   +   -   +
MD      +   +   +   +   +   -   +
BA      +   -   -   +   +   +   - ;

```

Un signo “+” indica un par que está en el conjunto, y un signo “-” indica que no está ese par. Normalmente las filas son etiquetadas con la primera componente, y las columnas con la segunda. Si preferimos lo opuesto, podemos indicar una tabla traspuesta al añadir (**tr**) después del nombre del conjunto:

```

set LINKS (tr):
           SE      MD      BA :=
CA        -       +       +
CO        +       +       -
HU        +       +       -

```

AL	-	+	+
JA	+	+	+
MA	-	-	+
GR	+	+	- ;

Las tablas son más convenientes para conjuntos que son relativamente densos. En otro caso la lista de pares va mejor; y es también más fácil de generar por un programa.

4. Otra forma de describir un conjunto de pares es listar todas las segundas componentes que unen con cada primera componente:

```

set LINKS :=
  (SE,*)      CO      HU      JA      GR
  (MD,*)      CA      CO      HU      AL      JA      GR
  (BA,*)      CA      AL      JA      MA ;

```

Se podría hacer listando todas las primeras componentes que unen con cada una de las segundas componentes: (*,CA) MD BA.

Cada comodín * es seguido por una lista, cuyas entradas son sustituidas por el * para generar pares en el conjunto.

Conjuntos multidimensionales.

Utilizamos los siguientes ejemplos, para ver las distintas formas de definir conjuntos multidimensionales:

```
set RUTAS within {ORIG,DEST,PROD};
```

```

set RUTAS :=
  (SE,HU,cuchillo) (SE,JA,cuchillo) (SE,GR,cuchillo)
  (MD,CA,plato) (MD,CA,cuchillo) (MD,CO,plato)
  ... ;

```

```

set RUTAS :=
  SE HU cuchillo SE JA cuchillo SE GR cuchillo
  MD CA plato MD CA cuchillo MD CO plato
  ... ;

```

```

set RUTAS :=
  (MD,*,plato) CA CO HU JA GR
  (BA,*,plato) CA AL JA MA
  ... ;

```

```

set RUTAS :=
  (*,CA,*) MD plato MD cuchillo BA plato
  (*,CO,*) MD plato MD cuchillo
  ... ;

```

```

set RUTAS :=
  (*,*,plato): CA CO HU AL JA MA GR :=
  SE      -      -      -      -      -      -
  MD      +      +      +      -      +      -      +
  BA      +      -      -      +      +      +      -

```

```

(*,*,cuchillo): CA  CO  HU  AL  JA  MA  GR  :=
SE      -      -      +      -      +      -      +
MD      +      +      +      +      +      -      -
BA      -      -      -      -      -      +      - ;

```

También se puede usar la notación (tr).

1.6.2 Datos de parámetros.

Para un parámetro escalar (no indexado), la asignación de un valor sería:

```
param avail := 40;
```

Parámetros unidimensionales.

La forma más simple para dar datos para un parámetro indexado es por medio de una lista. Para un parámetro indexado sobre un conjunto simple:

```
set PROD;
param valor {PROD} > 0;
```

Cada elemento de la lista de datos consta de un miembro del conjunto y de un valor:

```
set PROD:= plato cuchillo tenedor ;
param valor :=
    plato 200
    cuchillo 140
    tenedor 160 ;
```

equivalentemente:

```
param valor := plato 200, cuchillo 140, tenedor 160 ;
```

A menudo necesitamos datos para varios parámetros que están indexados sobre el mismo conjunto, en esta situación puede escribirse:

```
param valor := plato 200  cuchillo 140  tenedor 160 ;
param benefi := plato 25  cuchillo 30  tenedor 29 ;
param market := plato 6000  cuchillo 4000  tenedor 3500 ;
```

o

```
param:      valor      benefi  market  :=
plato      200        25        6000
cuchillo   140        30        4000
tenedor    160        29        3500 ;
```

Los dos puntos después de la palabra clave **param** es obligatoria; indica que damos valores para varios parámetros.

Si se sigue con el nombre del conjunto PROD y otros dos puntos:

```
param: PROD:      valor      benefi  market  :=
plato      200        25        6000
cuchillo   140        30        4000
tenedor    160        29        3500 ;
```

entonces los elementos del conjunto PROD son definidos también con este comando, evitando así la definición con el comando **set PROD**; el efecto es combinar las especificaciones del conjunto y los tres parámetros indexados sobre él.

Parámetros bidimensionales.

Los valores de un parámetro indexado sobre dos conjuntos, son generalmente introducidos como:

```

set ORIG;
set DEST;
param costo {ORIG,DEST} >= 0;

data
param costo: CA CO HU AL JA MA GR :=
SE      39  14  11  14 16  82  8
MD      27   9  12   9 26  95 17
BA      24  14  17  13 28  99 20 ;

```

Las etiquetas en las filas dan el primer índice y las etiquetas de la columna dan el segundo índice, así por ejemplo, `costo["SE","CA"]` se ha definido a 39.

Podemos usar la notación (`tr`):

```

param costo (tr):
           SE      MD      BA :=
CA      39      27      24
CO      14      9      14
... ;

```

Cuando son tablas grandes pueden utilizarse caracteres de nueva línea en cualquier lugar, o también emplear el siguiente formato:

```

param costo: CA CO HU AL :=
SE 39 14 11 14
MD 27 9 12 9
BA 24 14 17 13

: JA MA GR :=
SE 16 82 8
MD 26 95 17
BA 28 99 20 ;

```

Los dos puntos indica el comienzo de cada nueva subtabla; cada una tiene las mismas etiquetas de filas, pero diferentes etiquetas de columna.

El parámetro no tiene porque estar indexado sobre todas las combinaciones de miembros de `ORIG` y `DEST`, sino tan sólo de un subconjunto de esos pares:

```

set LINKS within {ORIG,DEST};
param costo {LINKS} >= 0;

```

Definidos como vimos en la sección anterior el conjunto `LINKS`, los valores de los parámetros podrían darse como:

```

param costo: CA CO HU AL JA MA GR :=
SE . 14 11 . 16 . 8
MD 27 9 12 9 26 . 17
BA 24 . . 13 28 99 . ;

```

Donde un “+” indica un miembro del conjunto, la tabla para `costo` da un valor. Donde un “-” indica que no pertenece, la tabla contiene un punto “.”. El punto puede aparecer en cualquier tabla `AMPL` para indicar “valor no especificado aquí”.

Podemos usar un símbolo diferente, por ejemplo `-`, al incluir el siguiente comando en `data`:

```

defaultsym "--";

```

Podemos desactivarlo al introducir el comando:

```

nodefaultsym ;

```

También podríamos introducir los datos del siguiente modo:

```

param costo :=
SE CO 14 SE HU 11 SE JA 16 SE GR 8

```



```
MD CA 27 ... ;
```

Cuando un parámetro está indexado sobre un subconjunto poco denso de pares, una lista puede ser más compacta y legible que la representación tabular, la cual estaría formada mayoritariamente por puntos. El formato de lista es también más fácil de programar para generarla con programas externos.

Otra ventaja del formato lista es que, como en el caso unidimensional, los datos para varias componentes pueden darse juntos:

```
param LINKS : costo limit :=
SE CO      14    1000
SE HU      11     800
SE JA      16    1200
SE GR       8    1100
MD CA      27    1200
MD CO       9     600
MD HU      12     900
MD AL       9     950
MD JA      26    1000
MD GR      17     800
BA CA      24    1500
BA AL      13    1400
BA JA      28    1500
BA MA      99    1200 ;
```

Esta tabla da simultáneamente los miembros de LINKS y los valores para costo, y también los valores para otros parámetros, limit, que está también indexado sobre LINKS.

Finalmente, la lista de datos para costo puede escribirse más concisamente al organizarla en trozos, como se mencionó para los miembros del conjunto LINKS en la sección previa.

```
set LINKS :=
(SE,*) CO HU JA GR
... ;

param costo :=
[SE,*) CO 14 HU 11 JA 16 GR 8
[MD,*) CA 27 CO 9 HU 12 AL 9 JA 26 GR 17
[BA,*) CA 24 AL 13 JA 28 MA 99 ;
```

Parámetros multidimensionales.

Podríamos introducirlos de las siguientes formas:

```
set ORIG;
set DEST;
set PROD;
set RUTAS within {ORIG,DEST,PROD};
param costo {RUTAS} >= 0;
```

lista simple

```
param costo :=
MD CO plato 9 MD CO cuchillo 8 MD CA plato 27
MD CA cuchillo 23 MD GR plato 17 ... ;
```

uso de trozos

```
param costo :=
```

```

[MD,*,plato] CA 27 CO 9 HU 12 JA 26 GR 17
[BA,*,plato] CA 24 AL 13 JA 28 MA 99
... ;
uso de trozos
param costo :=
  [*,CA,*] MD plato 27 MD cuchillo 23 BA plato 24
  [*,CO,*] MD plato 9 MD cuchillo 8
  ... ;
uso de trozos 2 dimensiones y tablas
param costo :=
  [*,*,plato]: CA CO HU AL JA MA GR :=
    MD 27 9 12 . 26 . 17
    BA 24 . . 13 28 99 .
  [*,*,cuchillo]: CA CO HU AL JA MA GR :=
    SE . . 11 . 16 . 8
    MD 23 8 10 9 21 . .
    BA . . . . . 81 . ;

```

Se puede emplear la notación (tr).

Otro ejemplo:

```

set PROD;
set AREA {PROD};
param T > 0;
param renta {p in PROD, AREA[p], 1..T} >= 0;

data

param T := 4;
set PROD := plato cuchillo ;
set AREA[plato] := este norte ;
set AREA[cuchillo] := este oeste export ;

param renta :=
[plato,*,*]: 1 2 3 4 :=
este 25.0 26.0 27.0 27.0
norte 26.5 27.5 28.0 28.5
[cuchillo,*,*]: 1 2 3 4 :=
este 30 35 37 39
oeste 29 32 33 35
export 25 25 25 28 ;

```

Valores por defecto.

AMPL comprueba que los comandos de datos asignan valores para exactamente todos los parámetros en el modelo. AMPL dará un mensaje de error si damos un valor para un parámetro inexistente, o nos olvidamos de dar un valor a un parámetro que existe.

Si el mismo valor apareciera muchas veces en un comando de datos, podemos especificar la frase **default**. Por ejemplo,

```

set ORIG;
set DEST;
set PROD;
param costo {ORIG,DEST,PROD} >= 0;

data

```

```

param costo default 9999 :=
  [*,*,plato]:  CA CO HU AL JA MA GR :=
                MD 27  9 12  . 26  . 17
                BA 24  .  . 13 28 99  .

  [*,*,cuchillo]: CA CO HU AL JA MA GR :=
                  SE  .  . 11  . 16  .  8
                  MD 23  8 10  9 21  .  .
                  BA  .  .  .  .  . 81  . ;

```

Tanto a los parámetros missing (como `costo["SE","CA","plato"]`), como a los marcados como omitidos con el uso de un punto (como `costo["SE","CA","cuchillo"]`), se les asignará el valor 9999. En total, hay 24 con valor 9999.

La característica **default** es especialmente útil cuando queremos que todos los parámetros de una colección indexada tengan el mismo valor. Por ejemplo:

```

param oferta {ORIG} >= 0;
param demanda {DEST} >= 0;

data

param oferta default 1 ;
param demanda default 1 ;

```

También, como se explicó en secciones anteriores, una declaración de parámetro puede incluir una expresión **default**. Por ejemplo:

```

param costo {ORIG,DEST,PROD} >= 0, default 9999;

```

Sin embargo, es mejor poner la frase **default** en los comandos de datos. La frase **default** debería ir en el modelo cuando queremos que el valor por defecto dependa de alguna forma de otros datos. Por ejemplo, un costo arbitrariamente grande podría darse para cada producto al especificar:

```

param gran_costo {PROD} > 0;
param costo {ORIG,DEST, p in PROD} >= 0, default gran_costo[p];

```

1.6.3 Datos para variables y restricciones.

Opcionalmente podemos asignar valores iniciales a las variables o restricciones del modelo, usando cualquiera de las opciones para asignar valores a parámetros. El nombre de la variable almacena su valor, y el nombre de la restricción el valor de la variable dual asociada.

```

var Trans:  CA CO HU AL JA MA GR :=
            SE 100 100 800 100 100 500 200
            MD 900 100 100 500 500 200 200
            BA 100 900 100 500 100 900 200 ;

```

También con una tabla simple podemos dar valores a parámetros (`valor`, `benefi`, `market`) y variables (`Make`):

```

param:      valor benefi market Make :=
plato       200   25   6000  3000
cuchillo    140   30   4000  2500
tenedor     160   29   3500  1500 ;

```

Los valores iniciales de las variables (o expresiones que envuelven esos valores iniciales) pueden verse antes de escribir **solve**, usando los comandos **display**, **print** o **printf**.

El uso más común de asignar valores iniciales a variables o restricciones es dar un punto de arranque para resolver un problema de optimización no lineal.

1.7 Comandos del lenguaje.

La llamada a AMPL normalmente causa la entrada en un entorno de comandos, donde los comandos pueden ser introducidos interactivamente. Las declaraciones del modelo y las instrucciones de introducción de datos son también aceptadas como comandos.

El entorno de comandos de AMPL reconoce dos modos. En modo **modelo**, reconoce las declaraciones del modelo y todos los comandos que se describirán a continuación. El otro modo es el modo **datos**, en el cual sólo se reconocen instrucciones referentes a la introducción de datos. El entorno siempre vuelve al modo modelo al encontrar una palabra clave que no comience con la palabra **data**.

Una frase de la forma:

```
include fichero;
```

trae el contenido del fichero al entorno de comandos de AMPL. Los comandos **include** pueden estar anidados, ellos son reconocidos en modo modelo y en modo datos.

Las secuencias:

```
model; include fichero;
```

```
data; include fichero;
```

pueden abreviarse como:

```
model fichero;
```

```
data fichero;
```

Los comandos no son parte de un modelo, pero producen que AMPL actúe como se describe en la tabla 7. Los comandos distintos a **data**, **end**, **include**, **quit** y **shell** producen que AMPL entre en modo modelo.

Desde la línea de comandos de AMPL podemos escribir, por ejemplo:

```
ampl: include ejemplo1.run;
```

siendo el fichero ejemplo1.run (tabla 8), un fichero por lotes que almacena la secuencia de comandos necesarios para resolver el ejemplo 1.1.

```
# EJEMPLO1.RUN
option solver cplex;
model ejemplo1.mod;
data ejemplo1.dat;
solve;
display z;
display x;
display restriccion.slack;
```

Tabla 8: Fichero de lotes para el modelo del ejemplo 1.1.

Los comandos **display**, **print** y **printf** imprimen expresiones arbitrarias. Tienen el siguiente formato:

```
display [conjunto: ] lista--argumentos [redireccion];
```

```
print [conjunto: ] lista--argumentos [redireccion];
```

```
printf [conjunto: ] fmt, lista--argumentos [redireccion];
```

Comandos	Significado
break	termina un bucle for o while
close	cierra un fichero
continue	salta al final del cuerpo del bucle
data	cambia a modo datos; opcionalmente incluye un fichero
display	imprime elementos del modelo y expresiones
delete	elimina un componente previamente declarado
drop	elimina una restricción u objetivo
end	finaliza la entrada del fichero de entrada actual
expand	muestra explícitamente el modelo
fix	fija una variable a su valor actual
for {indx}{ cp }	bucle for
if lexpr then { }	comprueba una condición
include	incluye ficheros
let	cambia los valores de los datos (:=)
match(cad,mod)	posición de mod en cad
model	cambia al modo modelo; opcionalmente incluye un fichero
objective	selecciona un objetivo a optimizar
option	define o muestra valores opcionales
print	imprime elementos del modelo sin formatear
printf	imprime elementos del modelo formateados ($\backslash n, \%7.1f$)
problem nb: def.	define un problema
purge	elimina un componente y los dependientes de él
quit	termina AMPL
read	lee datos de un fichero o de la consola (i-)
redeclare	redefine un componente ya definido
repeat while lexpr { cp }	repite un bloque de comandos mientras V.
repeat until lexpr { cp }	repite un bloque de comandos hasta F.
repeat { cp } while lexpr	repite un bloque de comandos mientras V.
repeat { cp } until lexpr	repite un bloque de comandos hasta F.
reset	resetea elementos específicos a su estado inicial
restore	deshace un comando drop
shell	temporalmente sale al sistema operativo
show	muestra componentes del modelo
solution	importa valores de variables de un solver
solve	envía elementos actuales a un solver y devuelve la solución
step <i>n</i>	avanza <i>n</i> pasos en la ejecución de ficheros por lotes
update	permite actualizar datos
unfix	deshace un comando fix
write	escribe en un fichero partes de un problema
xref	muestra dependencias del modelo

Tabla 7: Comandos del entorno AMPL.

Si el conjunto está presente, su campo de acción se extiende hasta el final del comando, y causa una ejecución del comando para cada miembro del conjunto. La cadena de formato `fmt` es como en el lenguaje C.

La lista-argumentos es una lista de expresiones separadas por comas.

El opcional `redireccin` tiene una de las dos formas siguientes:

```
> fichero
>> fichero
```

La primera abre por primera vez un fichero para escribir, y la segunda añade al fichero ya creado, aunque `>` actúa igual que `>>`, si el fichero está ya abierto.

Con el comando `option` se puede conseguir que la salida que se ha solicitado tenga un formato específico. Por ejemplo:

```
option display_precision 3;
option omit_zero_rows 1;
```

La primera especifica la precisión de salida (0 equivale a ninguna) y la segunda omite las salidas con valor cero (por defecto es 0, es decir no omite los valores cero).

Otras opciones son:

```
option solver_msg 0;
option relax_integrality 1;
option presolve 0;
option show_stats 1;
option times 1;
option gentimes 1;
option log_file 'hola.tmp';
option solution_round 6;
option single_step 1;
```

1.8 Optimizadores.

Ampl clasifica los optimizadores en diferentes tipos atendiendo al tipo de problema que resuelven y los métodos que usan, estos son:

- **Lineal (simplex)**: objetivo y restricciones lineales, resueltos con alguna versión del método del simplex.
- **Lineal (interior)**: objetivo y restricciones lineales, resueltos con alguna versión de métodos de punto interior (o barrera).
- **Redes**: objetivo lineal y restricciones de flujo en redes, resuelto por alguna versión del método simplex para redes.
- **Cuadrático**: objetivos cuadráticos convexos o cóncavos y restricciones lineales, resuelto por un método tipo simplex o método tipo punto interior.
- **Convexo**: objetivo convexo o cóncavo pero no todos lineales, y restricciones lineales, resuelto por un método de punto interior.
- **No lineal**: continuo pero no todos los objetivos y restricciones lineales, resuelto por cualquiera de los métodos siguientes: gradiente reducido, cuasi-newton, lagrangiano aumentado y punto interior.

- **Complementariedad:** Lineal o no lineal como antes, pero con condiciones de complementariedad adicional.
- **Lineal entero:** objetivo y restricciones lineales y alguna o todas las variables enteras, resuelto por alguna aproximación de ramificación y acotación que aplica un algoritmo lineal para resolver los subproblemas sucesivos.
- **Entero no lineal:** objetivos y restricciones continuos pero no todos lineales y alguna o todas las variables enteras, resuelto por la aproximación de ramificación y acotación que aplica un algoritmo no lineal para resolver los subproblemas sucesivos.

La siguiente tabla recoge los optimizadores conocidos por la versión estudiante de AMPL Plus:

Optimizador	Algoritmos	Versión AMPL
CPLEX	lineal (símplex) lineal (interior) redes cuadrático entero lineal	AMPL PLUS AMPL dos
XLSOL	lineal (símplex) cuadrático entero lineal	AMPL PLUS
MINOS	lineal (símplex) No lineal	AMPL PLUS AMPL dos
GRG2	No lineal Entero no Lineal	AMPL PLUS

Otros optimizadores que están implementados para usar con ampl son:

Optimizador	Algoritmos
LOQO	lineal (interior) cuadrático No lineal
LP_SOLVE	lineal (símplex) entero lineal
DONLP2	No lineal

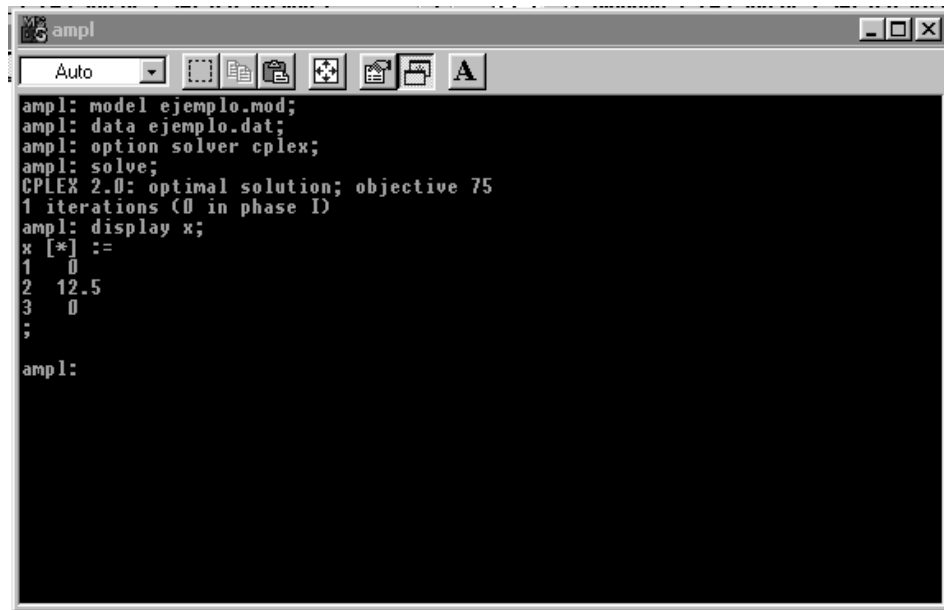
2 Entorno AMPL para Msdos.

AMPL para Msdos (o UNIX), opera a través de la “línea de comandos”, en la cual tenemos que ir introduciendo las instrucciones línea a línea.

Desde el sistema operativo podemos acceder a este entorno llamando al programa:

`ampl.exe`

Aparece la ventana de la figura 1:



```
ampl: model ejemplo.mod;
ampl: data ejemplo.dat;
ampl: option solver cplex;
ampl: solve;
CPLEX 2.0: optimal solution; objective 75
1 iterations (0 in phase I)
ampl: display x;
x [*] :=
1 0
2 12.5
3 0
;
ampl:
```

Figura 1: Entorno de comandos de AMPL para msdos: **ampl**.

Cuando aparece el prompt **ampl:** podemos introducir las instrucciones del lenguaje de modelado y los datos con la sintaxis de AMPL, y también cualquiera de los comandos vistos en la tabla 7, tales como: **reset**, **option** o **solve**.

Sin embargo este entorno de comandos AMPL no tiene facilidades para editar el modelo o los datos, o leer y escribir datos de /a un fichero fuente externo, para ello debemos ayudarnos de cualquier editor de código ASCII (por ejemplo el editor de Msdos: EDIT).

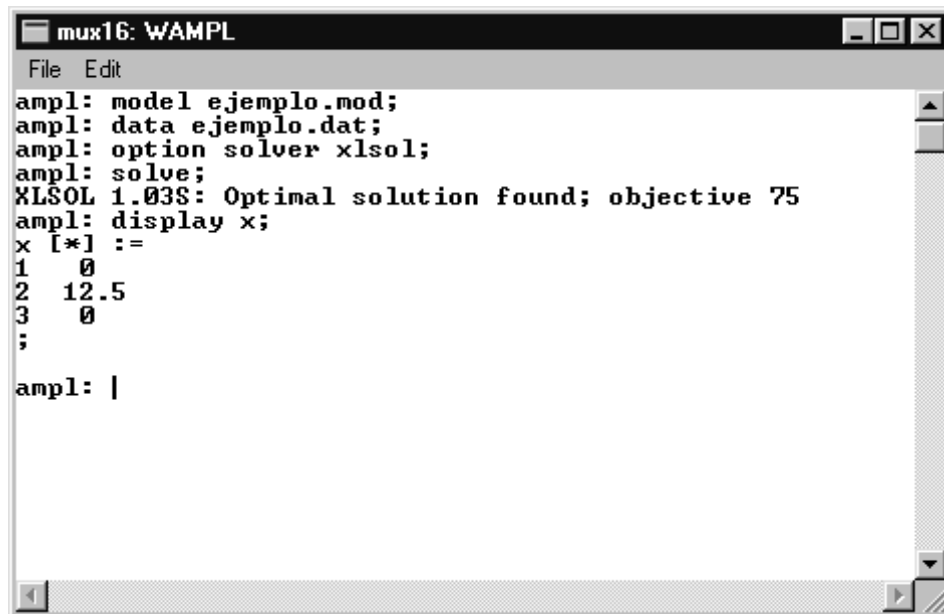
3 Entorno AMPL para Windows: AMPL Plus.

En **AMPL Plus**, el procesador del lenguaje AMPL es: **wampl.exe**, el cual permite trabajar con todas las capacidades del lenguaje AMPL, de la misma manera que el entorno AMPL para Msdos.

Desde el sistema operativo se puede acceder al procesador de comandos de forma independiente, llamando al programa:

wampl.exe

Aparece la ventana de la figura 2:



```
File Edit
ampl: model ejemplo.mod;
ampl: data ejemplo.dat;
ampl: option solver xlsol;
ampl: solve;
XLSOL 1.03S: Optimal solution found; objective 75
ampl: display x;
x [*] :=
1  0
2 12.5
3  0
;
ampl: |
```

Figura 2: Entorno de comandos de AMPL Plus: **wampl**.

Se puede apreciar en la figura, como se interactúa desde este programa para resolver un problema modelado con AMPL.

Desde este programa es posible también ejecutar ficheros por lotes (con extensión: **run**) con el comando “**include ejemplo.run;**”, o con el comando “**commands ejemplo.run;**”.

3.1 Componentes de AMPL Plus.

En la versión de AMPL Plus para Windows con entorno gráfico se usan los mismos comandos que deberíamos escribir en AMPL estándar, pero los comandos están normalmente representados bien por menús o por botones de acceso rápido. Desde este entorno, podemos inspeccionar los comandos generados automáticamente o las respuestas, escribir los comandos estándar de AMPL, examinar, imprimir o grabar los resultados, de una manera muy sencilla.

El sistema de modelación de AMPL Plus consta de cuatro componentes principalmente:

- El interface gráfico del usuario,
- el procesador del lenguaje AMPL,
- los optimizadores, y
- los drivers de ODBC para manejo de bases de datos.

En la siguiente figura podemos ver las características principales del entorno de AMPL Plus:

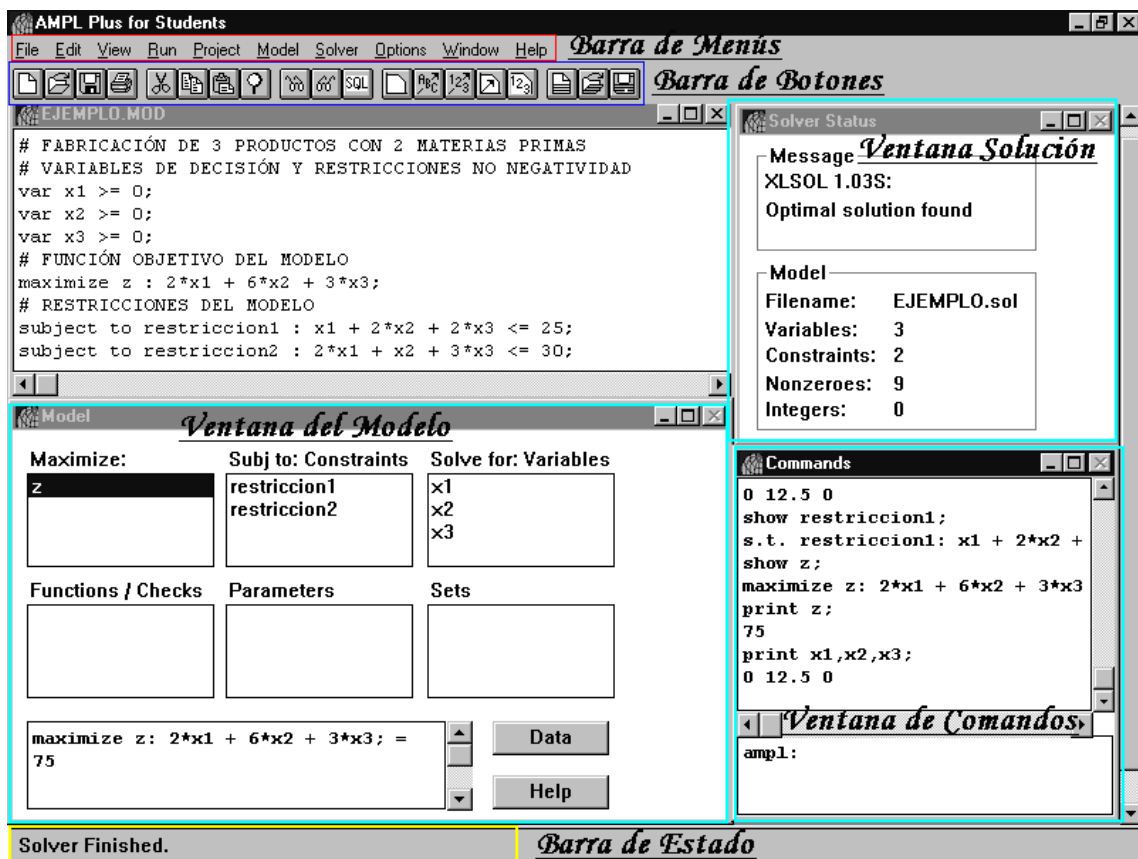


Figura 3: Interface de AMPL Plus.

En la ventana principal de AMPL Plus se pueden apreciar los siguientes componentes:

- **La barra de Menús.** Nos permite acceder a todos los comandos necesarios para controlar AMPL Plus. La mayoría de las opciones de menú tienen aceleradores de teclado. Por ejemplo para resolver el problema del ejemplo 1.1, una vez escrito el fichero: EJEMPLO.MOD (aparece en la tabla 1), pulsamos **F7** (Run->Build Model) y a continuación pulsamos **F9** (Run->Solve Problem).
- **La barra de Botones.** Nos permite acceder con un solo click de ratón a las tareas más comunes de la barra de menús.
- **La barra de Estado.** Muestra un pequeño mensaje sobre la operación más recientemente realizada. En esta pantalla ejemplo, aparece: **Solver Finished**, lo que nos indica que el problema ha sido resuelto.
- **La ventana de Comandos (Commands).** Se divide en dos áreas: *área (panel inferior) de entrada de comandos de AMPL estándar* (tales como `display`, etc) y *un área (panel superior) donde se visualizan los resultados de aplicar los comandos introducidos*.

Los mensajes aparecen clasificados en varios tipos: (1) los mismos comandos, (2) los mensajes de error, (3) salidas y (4) mensajes del optimizador o algoritmo de resolución.

- **La ventana de Estado de la Solución (Solver Status).** Muestra información sobre la operación realizada más recientemente en el Menú Run, tal como: *construir un fichero de modelo o un fichero de datos, o resolver un problema*. Cuando un problema es resuelto muestra información del problema (tal como el número variables y restricciones) y el progreso de la solución (tal como el valor actual de la función objetivo). También existen botones para detener momentáneamente (**pause**) o definitivamente (**stop**) la resolución del problema.
- **La ventana del Modelo (Models).** Después de que uno o más ficheros de modelos se han dirigido al procesador del lenguaje AMPL, esta ventana puede usarse para examinar el estado del modelo construido por AMPL.
Se muestran varias listas que contienen los nombres de los elementos del modelo (tales como los nombres de conjuntos, parámetros, variables, objetivos y restricciones) que AMPL ha reconocido. Cuando hacemos click con el ratón sobre un nombre, aparece su definición. Si están disponibles los valores de los datos al hacer click sobre el botón de datos (**Data**) se crea una ventana en la que podemos mostrar los resultados seleccionados en forma de hoja de cálculo.
- **Las ventanas creadas por el usuario** (en la figura ejemplo aparece editado el fichero EJEMPLO.MOD). Son ventanas de edición en las que podemos crear nuevos ficheros o abrir ficheros ya creados, de dos tipos: **para editar texto ASCII** (generalmente los ficheros relacionados con el problema: modelo, datos o fichero de ejecución de lotes) y **ficheros de datos** (extensión .qry).

3.2 Resolución de problemas en AMPL Plus.

Los optimizadores o programas de resolución en AMPL Plus (DLLs) pueden encontrar la solución mientras continuamos haciendo otras tareas y además podemos seguir la evolución de la resolución del problema, mientras que en la versión estándar de Msdos no es posible.

Cuando un programa de resolución está trabajando en optimizar un problema, lee un fichero de problema generado por AMPL (por ejemplo: problema.nl) y, cuando finaliza, escribe un fichero solución (por ejemplo: problema.sol) el cual es leído por el procesador del lenguaje AMPL. Los valores solución para las variables de decisión son usadas para actualizar el resto de valores en el problema, tales como los valores calculados de la función objetivo y de las restricciones. Todos estos valores los podemos visualizar bien con ayuda de los comandos estándar de AMPL (**display** en la ventana de comandos en AMPL Plus) o a través de las ventanas de visualización del Modelo de AMPL Plus.

Los modos más habituales de resolver problemas de optimización con AMPL Plus son:

1. Le pedimos a AMPL Plus usando el menú: **Run**, que construya por un lado el fichero del modelo (submenú: **Build Model** o **F7**), luego el fichero de datos (submenú: **Build Data** o **F8**), y a continuación si no hubo ningún error, llamar al optimizador (submenú: **Solve Problem** o **F9**). Para obtener un resumen de la solución podemos llamar al submenú: **Build Results** o directamente pulsar **F10**.
2. Se puede crear un **proyecto** (Menú de proyectos) indicando quién es el fichero del modelo (extensión .mod) el fichero de datos (extensión .dat o .qry) un fichero de lotes (extensión .run, que no contenga comandos de lectura de ficheros de modelos y datos) donde se indican algunas opciones (en realidad podemos introducir varios ficheros de lotes en un solo proyecto) y salidas particulares, y por último puede ir un fichero donde se escribirán los resultados del problema (extensión habituales .dat, .qry, ó .sal), pero éste debe aparecer abierto en AMPL Plus antes de ser resuelto.

Podemos ver este proceso en la figura 4.

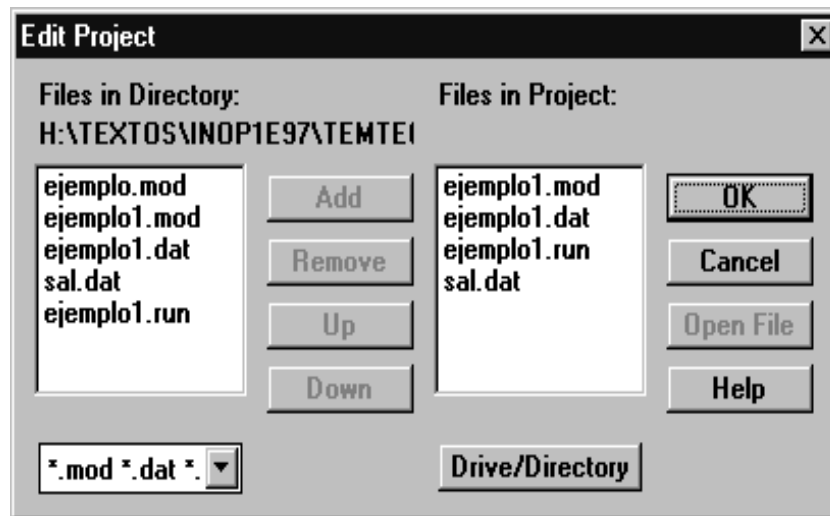


Figura 4: Edición de un proyecto de AMPL Plus.

El proyecto se puede guardar (recibirá extensión: `.amp`), y para resolver el problema asociado a él tan sólo es necesario que le pidamos que construya los resultados (**F10**).

También es posible usar este fichero de proyecto para resolver el problema desde la línea de comandos del sistema operativo (sin necesidad de abrir el programa AMPL Plus) de la siguiente forma:

```
amplplus ejemplo.amp
```

En este caso caso no debería aparecer un fichero para la recogida de datos (ya que no podría estar abierto previamente).

3. Si creamos un fichero por lotes (suelen tener extensión: **run**) que contenga todos los comandos necesarios para leer el modelo, leer los datos y otros comandos del lenguaje ampl (ver tabla 7), y ejecutarlo desde la ventana de comandos (por ejemplo: **include camino\ejemplo1.run**).

Nota: deben de aparecer los caminos exactos de donde se encuentras los ficheros a leer, incluido el fichero de lotes.